# 套接字

## 12.1引言

前面讨论了了各种LINUX/UNIX系统所提供的经典进程间通信（IPC）机制：管道、先进先出、消息队列、信号量以及共享内存。通过这些机制，同一台计算机上运行的进程可以相互通信。本章将讨论不同计算机（通过网络相连）上运行的进程相互通信的机制：网络进程间通信（network IPC）。

在本章中，将描述套接字网络IPC接口，进程能够使用该接口和其他进程通信。通过该接口，其他进程运行位置是透明的，它们可以在同一台计算机上也可以在不同的计算机上。实际上，这正是套接字接口的目标之一：同样的接口既可以用于计算机间通信又可以用于计算机内通信。尽管套接字接口可以采用许多不同的网络协议，但本章的讨论仅限于因特网事实上的通信标准：TCP/IP协议栈。

## 12.2套接字描述符

套接字是通信端点的抽象。与应用程序要使用文件描述符访问文件一样，访问套接字也需要用套接字描述符。套接字描述符在LINUX/UNIX系统是用文件描述符实现的。事实上，许多处理文件描述符的函数（如read和write）都可以处理套接字描述符。

要创建一个套接字，可以调用socket函数。

|  |
| --- |
| #include <sys/socket.h>  int socket (int domain, int type, int protocol);  返回值：若成功则返回文件(套接字)描述符，若出错则返回-1 |

参数domain （域）确定通信的特性，包括地址格式（在下一小节详细讲述）。下表总结了常见的各个域。各个域有自己的格式表示地址，而表示各个域的常数都以AF\_开头，意指地址族（address family）。

|  |  |
| --- | --- |
| 套接字通信域 | |
| 域 | 描述 |
| AF\_INET | IPv4因特网域 |
| AF\_INET6 | IPv6因特网域 |
| AF\_UNIX | UNIX域 |
| AF\_UNSPEC | 未指定 |

UNIX域这里不做讨论。多数系统还会定义AF\_L0CAL域，这是AF\_UNIX的别名。 AF\_UNSPEC域可以代表任何域。历史上，有些平台支持其他网络协议（如AF\_IPX为NetWare 协议族）。

参数type确定套接字的类型，进一步确定通信特征。下表总结了常见的套接字类型，但在实现中可以自由增加对其他类型的支持。

|  |  |
| --- | --- |
| 套接字类型 | |
| 类型 | 描述 |
| S0CK\_DGRAM | 长度固定的、无连接的不可靠报文传递 |
| S0CK\_RAW | IP协议的数据报接口 |
| SOCK\_SEQPACKET | 长度固定、有序、可靠的面向连接报文传递 |
| S0CK\_STREAM | 有序、可靠、双向的面向连接字节流 |

参数protocol通常是零，表示按给定的域和套接字类型选择默认协议。当对同一域和套接字类型支持多个协议时，可以使用protocol参数选择一个特定协议。在AF\_INET通信域中套接字类型SOCK\_STREAM的默认协议是TCP（传输控制协议）。在AF\_INET通信域中套接字类型SOCK\_DGRAM的默认协议是UDP （用户数据报协议）。

对于数据报（SOCK\_DGRAM）接口，与对方通信时是不需要逻辑连接的。只需要送出一个报文，其地址是一个对方进程所使用的套接字。

因此数据报提供了一个无连接的服务。另一方面，字节流（SOCK\_STREAM）要求在交换数据之前，在本地套接字和与之通信的远程套接字之间建立一个逻辑连接。

数据报是一种自包含报文。发送数据报近似于给某人邮寄信件。可以邮寄很多信，但不能保证投递的次序，并且可能有些信件丢失在路上。每封信件包含接收者的地址，使这封信件独立于所有其他信件。每封信件可能送达不同的接收者。

相比之下，使用面向连接的协议通信就像与对方打电话。首先，需要通过电话建立一个连接，连接建立好之后，彼此能双向地通信。每个连接是端到端的通信信道。会话中不包含地址信息，就像呼叫的两端存在一个点对点虚拟连接，并且连接本身暗含特定的源和目的地。

对于SOCK\_STREAM套接字，应用程序意识不到报文界限，因为套接字提供的是字节流服务。这意味着当从套接字读出数据时，它也许不会返回所有由发送进程所写的字节数。最终可以获得发送过来的所有数据，但也许要通过若干次函数调用得到。

SOCK\_SEQPACKET套接字和SOCK\_STREAM套接字很类似，但从该套接字得到的是基于报文的服务而不是字节流服务。这意味着从SOCK\_SEQPACKET套接字接收的数据量与对方所发送的一致。流控制传输协议（Stream Control Transmission Protocol, SCTP）提供了因特网域上的顺序数据包服务。

SOCK\_RAW套接字提供一个数据报接口用于直接访问下面的网络层（在因特网域中为BP）。使用这个接口时，应用程序负责构造自己的协议首部，这是因为传输协议（TCP和UDP等）被绕过了。 当创建一个原始套接字时需要有超级用户特权，用以防止恶意程序绕过内建安全机制来创建报文。

调用socket与调用open相类似。在两种情况下，均可获得用于输入/输出的文件描述符。

当不再需要该文件描述符时，调用close来关闭对文件或套接字的访问，并且释放该描述符以便重新使用。

虽然套接字描述符本质上是一个文件描述符，但不是所有参数为文件描述符的函数都可以接受套接字描述符。下表总结了到目前为止所讨论的大多数使用文件描述符的函数处理套接字描述符时的行为。未规定的和由实现定义的行为通常意味着函数不能处理套接字描述符。例如，lseek不处理套接字，因为套接字不支持文件偏移量的概念。

|  |  |
| --- | --- |
| 使用文件描述符的函数处理套接字时的行为 | |
| 函数 | 处理套接字时的行为 |
| close | 释放套接字 |
| dup, dup2 | 和一般文件描述符一样复制 |
| fchdir | 失败，并且将errno设置为EN0TDIR |
| fchmod | 未规定 |
| fchown | 由实现定义 |
| fcntl | 支持一些命令，例如F\_DUPFD，F\_GETFD, F\_GETFL，F\_GET0WN，F\_SETFD，F\_SETFL，F\_SET0WN |
| fdatasync, fsync | 由实现定义 |
| fstat | 支持一些stat结构成员，但如何支持由实现定义 |
| ftruncate | 未规定 |
| ioctl | 支持一些命令，依赖于底层驱动 |
| lseek | 由实现定义（通常是失败并且将errno设为ESPIPE） |
| read | 与没有任何标志位的recv等价 |
| write | 与没有任何标志位的send等价 |

套接字通信是双向的。可以采用函数shutdown来禁止套接字上的输入输出。

|  |
| --- |
| #include <sys/socket,h>  int shutdown (int sockfd, int how)；  返回值：若成功则返回0，若出错则返回-1 |

如果how是SHUT\_RD （关闭读端），那么无法从套接字读取数据，如果how是SHUT\_WR（关闭写端），那么无法使用套接字发送数据，使用SHUT\_RDWR则将同时无法读取和发送数据。

能够close（关闭）套接字，为何还使用shutdown呢？理由如下：首先，close只有在最后一个活动引用被关闭时才释放网络端点。这意味着如果复制一个套接字（例如采用dup），套接字直到关闭了最后一个引用它的文件描述符之后才会被释放。而shutdown允许使一个套接字处于不活动状态，无论引用它的文件描述符数目多少。其次，有时只关闭套接字双向传输中的一个方向会很方便。例如，如果想让所通信的进程能够确定数据发送何时结束，可以关闭该套接字的写端，然而通过该套接字读端仍可以继续接收数据。

## 12.3寻址

上一节中学习了如何创建和销毁一个套接字。在学习用套接字做一些有意义的事情之前，需要知道如何确定一个目标通信进程。进程的标识有两个部分：计算机的网络地址可以帮助标识网络上想与之通信的计算机，而服务可以帮助标识计算机上特定的进程。

### 12.3.1字节序

运行在同一台计算机上的进程相互通信时，一般不用考虑字节的顺序（字节序），字节序是一个处理器架构特性，用于指示像整数这样的大数据类型的内部字节顺序。下图显示一个32位整数内部的字节常用的两种排序方式：大端字节序和小端字节序。

|  |
| --- |
|  |
| 大段字节序和小端字节序 |

如果处理器架构支持大端（big-endian）字节序，那么最大字节地址对应于数字最低有效字节（LSB）上，小端（little-endian）字节序则相反：数字最低字节对应于最小字节地址。注意，不管字节如何排序，数字最高位总是在左边，最低位总是在右边。因此，如果想给一个32位整数赋值0x04030201，不管字节如何排序，数字最高位包含4，数字最低位包含1。如果接着想将一个字符指针（cp）强制转换到这个整数的地址，将看到字节序带来的不同。在小端字节序的处理器上，cp[0]指向数字最低位随包含1，cp[3]指向数字最高位随包含4。相比较而言，对于大端字节序的处理器，cp[0]指向数字最高位因而包含4，cp[3]指向数字最低位因而包含1。有些处理器（CPU）可以配置成大端或小端，使问题变得乱上加乱。

网络协议指定了字节序，因此异构计算机系统能够交换协议信息而不会混淆字节序。 TCP/IP协议栈采用大端字节序。应用程序交换格式化数据时，字节序问题就会出现。对于 TCP/IP，地址用网络字节序来表示，所以应用程序有时需要在处理器的字节序与网络字节序之间的转换。例如，当打印一个易于阅读的地址时，这种转换是很平常的。

对于TCP/IP应用程序，提供了四个通用函数以实施在处理器字节序和网络字节序之间的转换。

|  |
| --- |
| #include <arpa/inet.h>  uint32\_t htonl (uint32\_t hostint32)；  返回值：以网络字节序表示的32位整型数  uintl6\_t htons (uintl6\_t hostintl6);  返回值：以网络字节序表示的16位整型数  uint32\_t ntohl (uint32\_t netint32)；  返回值：以主机字节序表示的32位整型数  uintl6\_t ntohs (uintl6\_t netintl6)；  返回值：以主机字节序表示的16位整型数 |

h表示“主机（host）”字节序，n表示“网络（network）”字节序。1表示“长（long）”整数（即4个字节），s表示“短（short）”整数（即2个字节）。这4个函数定义在<arpa/inet.h>中, 也有比较老的系统将其定义在<netinet/in.h>中。

### 12.3.2地址格式

地址标识了特定通信域中的套接字端点，地址格式与特定的通信域相关。为使不同格式地址能够被传入到套接字函数，地址被强制转换成通用的地址结构sockaddr表示：

|  |
| --- |
| struct sockaddr {  sa\_family\_t sa\_family; /\* address family \*/  char sa\_data [] /\* variable-length address \*/  .  .  } |

套接字实现可以自由地添加额外的成员并且定义sa\_data成员的大小。例如在Linux中，该结构定义如下：

struct sockaddr {

sa\_family\_t sa\_family; /\* address family \*/

char sa\_data[14]; /\* variable-length address \*/

};

因特网地址定义在<netinet/in.h>中。在IPv4因特网域（AF\_INET）中，套接字地址用如下结构sockaddr\_in表示：

struct in\_addr {

in\_addr\_t s\_addr; /\* IPv4 address \*/

struct sockaddr\_in {

sa\_family\_t sin\_family;

in\_port\_t sin\_port;

struct in\_addr sin\_addr;

}

数据类型in\_port\_t定义成uintl6\_t。数据类型in\_addr\_t定义成uint32\_t。这些整数类型在<stdint.h>定义并指定了相应的位数。

这些是Single UNIX Specification必需的定义，每个实现可以自由地添加额外的字段。例如，在Linux中，sockaddr\_in定义如下：

struct sockaddr\_in {

sa\_family\_t sin\_family; /\* address family \*/

in\_port\_t sin\_port; /\* port number \*/

struct in\_addr sin\_addr; /\* Pv4 address \*/

unsigned char sin\_zero[8]; /\* filler \*/

}

其中成员sin\_zero为填充字段，必须全部被置为0。

有时，需要打印出能被人而不是计算机所理解的地址格式。LINUX/UNIX中包含了函数 inet\_addr和inet\_ntoa，用于在二进制地址格式与点分十进制字符串表示（a.b.c.d）之间相互转换。这些函数仅用IPv4地址，但功能相似的两个新函数inet\_ntop和inet\_pton支持IPv4和IPv6地址。

|  |
| --- |
| #include <arpa/inet.h>  const char \*inet\_ntop(int domain, const void\* restrict addr, char\* restrict str, socklen\_t size);  返回值：若成功则返回地址字符串指针，若出错则返回NULL  int inet\_pton (int domain, const char\* restrict str, void\* restrict addr);  返回值：若成功则返回1，若格式无效则返回0，若出错则返回-1 |

函数inet\_ntop将网络字节序的二进制地址转换成文本字符串格式，inet\_pton将文本字符串格式转换成网络字节序的二进制地址。参数仅支持两个值：AF\_INET和AF\_ INET6。

对于inet\_ntop，参数size指定了用以保存文本字符串的缓冲区（str）的大小。两个常数用于简化工作：INET\_ADDRSTRLEN定义了足够大的空间来存放表示IPv4地址的文本字符串，INET6\_ADDRSTRLEN定义了足够大的空间来存放表示IPv6地址的文本字符串。对于inet\_pton，如果domain是AF\_INET，缓冲区addr需要有足够大的空间来存放32位地址，如果domain为AF\_INET6则需要足够大的空间来存放128位地址。

### 12.3.3地址查询

理想情况下，应用程序不需要了解套接字地址的内部结构。如果应用程序只是简单地传递类似于sockaddr结构的套接字地址，并且不依赖于任何协议相关的特性，那么可以与提供相同服务的许多不同协议协作。

在本节，将讨论一些细节，并且引入新的函数来査询寻址信息。这些函数返回的网络配置信息可能存放在许多地方。它们可以保存在静态文件中（如, /etc/hosts，/etc/services等），或者可以由命名服务管理，例如DNS（Domain Name System）或者NIS（Network Information Service）。无论这些信息放在何处，这些函数同样能够访问它们。

通过调用gethostent，可以找到给定计算机的主机信息。

|  |
| --- |
| #include <netdb.h>  struct hostent＊ gethostent(void)；  返回值：若成功则返回指针，若出错则返回NULL  void sethostent (int stayopen);  void endhostent(void); |

如果主机数据文件没有打开，gethostent会打开它。函数gethostent返回文件的下一个条目。函数sethostent会打开文件，如果文件已经被打开，那么将其回绕。函数endhostent 将关闭文件。

当gethostent返回时，得到一个指向hostent结构的指针，该结构可能包含一个静态的数据缓冲区。每次调用gethostent将会覆盖这个缓冲区。数据结构hostent至少包含如下成员：

struct hostent

{

|  |  |
| --- | --- |
| \*h\_name； | /\* name of host \*/ |
| \*\*h\_aliases; | /\* pointer to alternate host name array \*/ |
| h \_addrtype； | /\* address type \*/ |
| h\_length; | /\* length in bytes of address \*/ |
| \*\*h \_addr list； | /\* pointer to array of network addresses \*/ |

}

返回的地址采用网络字节序。

两个附加的函数gethostbyname和gethostbyaddr，原来包含在hostent函数里面，现在被认为是过时的，马上将会看到其替代函数。

能够采用一套相似的接口来获得网络名字和网络号。

|  |
| --- |
| #include <netdb.h>  struct netent\* getnetbyaddr(uint32\_t net, int type);  struct netent\* getnetbyname(const char\* name);  struct netent\* getnetent(void);  以上三个函数的返回值：若成功则返回指针，若出错则返回null  void setnetent (int stayopen);  void endnetent(void); |

结构netent至少包含如下字段：

struct netent {

char \*n\_name; /\* official name of net \*/

char \*\*n\_aliases; /\* alias list \*/

int n\_addrtype; /\* net number type \*/

uint32\_t n\_net; /\* net number \*/

};

网络号按照网络字节序返回。地址类型是一个地址族常量（例如AF\_INET）。

可以将协议名字和协议号采用以下函数映射。

|  |
| --- |
| #include <netdb.h>  struct protoent \* get protobyname (const char \*name);  struct protoent \*getprotobynumber (int proto);  struct protoent \*getprotoent(void);  以上所有函数的返回值：若成功则返回指针，若出错则返回NULL  void setprotoent (int stayopen);  void endprotoent(void); |

protoent至少包含如下成员;

struct protoent {

char \*p\_name; /\* official name of protocol \*/

char \*\*p\_aliases; /\* alias list \*/

int p\_proto; /\* protocol number \*/

};

服务是由地址的端口号部分表示的。每个服务由一个唯一的、熟知的端口号来提供。采用函数getservbyname可以将一个服务名字映射到一个端口号，函数getservbyport将一个端口号映射到一个服务名，或者采用函数getservent顺序扫描服务数据库。

|  |
| --- |
| #include <netdb.h>  struct servent \*getservbyname (const char \*name, const char\* proto);  struct servent \*getservbyport (int port, const char\* proto);  struct servent \*getservent(void);  以上所有函数的返回值：若成功则返回指针，若出错则返回NULL  void setservent (int stayopen);  void endservent(void); |

结构servent至少包含如下成员：

struct servent {

char \*s\_name; /\* official name of service \*/

char \*\*s\_aliases; /\* alias list \*/

int s\_port; /\* port service resides at \*/

char \*s\_proto; /\* protocol to use \*/

};

LINUX/UNIX提供了若干新的函数，允许应用程序将一个主机名字和服务名字映射到一个地址，或者相反。这些函数代替老的函数gethostbyname和gethostbyaddr。

函数getaddrinfo允许将一个主机名字和服务名字映射到一个地址。

|  |
| --- |
| #include <sys/sock:et ,h>  #include <netdb.h>  int getaddrinfo (const char\* restrict host,  const char\* re strict service,  const struct addrinfo\* restrict hint,  struct addrinfo\* restrict res)；  返回值：若成功则返回0，若出错则返回非0错误码  void freeaddrinfo (struct addrinfo \*ai)； |

需要提供主机名字、服务名字，或者两者都提供。如果仅仅提供一个名字，另外一个必须是一个空指针。主机名字可以是一个节点名或点分十进制记法表示的主机地址。

函数getaddrinfo返回一个结构addrinfo的链表。可以用freeaddrinfo来释放一个或多个这种结构，这取决于用ai\_next字段链接起来的结构有多少。

结构addrinfo的定义至少包含如下成员：

struct addrinfo {

int ai\_flags; /\* input flags \*/

int ai\_family; /\* protocol family for socket \*/

int ai\_socktype; /\* socket type \*/

int ai\_protocol; /\* protocol for socket \*/

socklen\_t ai\_addrlen; /\* length of socket-address \*/

struct sockaddr \*ai\_addr; /\* socket-address for socket \*/

char \*ai\_canonname; /\* canonical name for service location \*/

struct addrinfo \*ai\_next; /\* pointer to next in list \*/

};

根据某些规则，可以提供一个可选的hint来选择地址。是一个用于过滤地址的模板，仅使用ai\_family、ai\_flags、ai\_protocol和ai\_socktype字段。剰余的整数字段必须设为零，并且指针字段为空。下表总结了在ai\_flags中所用的标志，这些标志用来指定如何处理地址和名字。

|  |  |
| --- | --- |
| addrinfo结构标志 | |
| 标志 | 描述 |
| AI\_ADDRC0NFIG | 查询配置的地址类型（IPv4或IPv6） |
| AI\_ALL | 查找IPv4和IPv6地址（仅用于AI—V4MAPPED） |
| AI\_CAN0NNAME | 需要一个规范名（而不是别名） |
| AI\_NUMERICHOST | 以数字格式返回主机地址 |
| AI\_NUMERICSERV | 以端口号返回服务 |
| AI\_PASSIVE | 套接字地址用于监听绑定 |
| AI\_V4MAPPED | 如没找到IPv6地址，则返回映射到IPv6格式的IPv4地址 |

如果getaddrinfo失败，不能使用perror或strerror来生成错误消息。替代地，调用 gai\_strerror将返回的错误码转换成错误消息。

|  |
| --- |
| #include <netdb.h>  const char \*gai\_strerror(int error)；  返回值：指向描述错误的字符串的指针 |

函数getnameinfo将地址转换成主机名或者服务名。

|  |
| --- |
| #include <sys/types.h>  #include <sys/socket.h>  #include <netdb.h>  int getnameinfo(const struct sockaddr \*addr,  socklen\_t salen, char \*host,  socklen\_t hostlen, char \*service,  socklen\_t servlen, int flags);  返回值：若成功则返回0，若出错则返回非0值 |

套接字地址（addr、被转换成主机名或服务名。如果host非空，它指向一个长度为hostlen字节的缓冲区用于存储返回的主机名。同样，如果service非空，它指向一个长度为servlen字节的缓冲区用于存储返回的服务名。

参数flags指定一些转换的控制方式，下表总结了系统支持的标志。

|  |  |
| --- | --- |
| getnameinfo函数标志 | |
| 标志 | 描述 |
| NI\_DGRAM | 服务基于数据报而非基于流 |
| NI\_NAMEREQD | 如果找不到主机名字，将其作为一个错误对待 |
| NI\_N0FQDN | 对于本地主机，仅返回完全限定域名的节点名字部分 |
| NI\_NUMERICHOST | 以数字形式而非名字返回主机地址 |
| NI\_NUMERICSERV | 以数字形式而非名字返回服务地址（即端口号） |

下面程序演示了函数getaddrinfo的使用方法。

|  |
| --- |
|  |

程序运行结果如下：

|  |
| --- |
|  |

这个程序说明了函数getaddrinfo的使用方法。如果有多个协议为指定的主机提供相应的服务，程序会打印出超过一条的信息。本例中，仅仅打印工作在IPv4 (ai\_family为AF\_INET)上协议的地址信息。如果想将输出限制在AF\_INET协议族，可以在hint中设置 ai\_family 字段。

## 12.3.4将套接字与地址绑定

与客户端的套接字关联的地址没有太大意义，可以让系统选一个默认的地址。然而，对于服务器，需要给一个接收客户端请求的套接字绑定一个众所周知的地址。客户端应有一种方法来发现用以连接服务器的地址，最简单的方法就是为服务器保留一个地址并且在/etc/services或者某个名字服务（name service）中注册。

可以用bind函数将地址绑定到一个套接字。

|  |
| --- |
| #include <sys/socket.h>  int bind (int sockfd, const struct sockaddr \*addr, socklen\_t len);  返回值：若成功则返回0，若出错则返回-1 |

* 对于所能使用的地址有一些限制：
* 在进程所运行的机器上，指定的地址必须有效，不能指定一其他机器的地址。
* 地址必须和创建套接字时的地址族所支持的格式相匹配。
* 端口号必须不小于1024，除非该进程具有相应的特权(即为超级用户)。
* 一般只有套接字端点能够与地址绑定，尽管有些协议允许多重绑定。

对于因特网域，如果指定IP地址为INADDR\_ANY，套接字端点可以被绑定到所有的系统网络接口。这意味着可以收到这个系统所安装的所有网卡的数据包。在下一节中将看到，如果调用connect或listen，但没有绑定地址到一个套接字，系统会选一个地址并将其绑定到套接字。

可以调用函数getsockname来发现绑定到一个套接字的地址。

|  |
| --- |
| #include <sys/socket.h>  int getsockname (int sockfd, struct sockaddr ＊restrict addr,  socklen\_t ＊restrict alenp)；  返回值：若成功则返回0，若出错则返回-1 |

调用getsockname之前，设置为一个指向整数的指针，该整数指定缓冲区sockaddr的大小。返回时，该整数会被设置成返回地址的大小。如果该地址和提供的缓冲区长度不匹配, 则将其截断而不报错。如果当前没有绑定到该套接字的地址，其结果没有定义。

如果套接字已经和对方连接，调用getpeername来找到对方的地址。

|  |
| --- |
| #include <sys/socket.h>  int getpeername (int sochfd, struct sockaddr\* restrict addr, socklen\_t\* restrict alenp) ;  返回值：若成功则返回0,若出错则返回-1 |

除了还会返回对方的地址之外，函数getpeername和getsockname—样。

## 12.4建立连接

如果处理的是面向连接的网络服务（SOCK\_STREAM或SOCK\_SEQPACKET），在开始交换数据以前，需要在请求服务的进程套接字（客户端）和提供服务的进程套接字（服务器）之间建立一个连接。可以用connect建立一个连接。

|  |
| --- |
| #include <sys/socket,h>  int connect (int sockfd, const struct sockaddr \*addr, socklen\_t len);  返回值：若成功则返回0，若出错则返回-1 |

在connect中所指定的地址是想与之通信的服务器地址。如果没有绑定到一个地址，connec t会给调用者绑定一个默认地址。

当连接一个服务器时，出于一些原因，连接可能失败。要连接的机器必须开启并且正在运行，服务器必须绑定到一个想与之连接的地址，并且在服务器的等待连接队列中应有足够的空间（马上将学到这一点）。因此，应用程序必须能够处理connect返回的错误，这些错误可能由一些瞬时变化条件引起。

下面演示了一种如何处理瞬时connect错误的方法。这在一个负载很重的服务器上很有可能发生。

|  |
| --- |
|  |

运行结果如下：

|  |
| --- |
|  |

这个函数使用了名为指致补偿（exponential backoff）的算法。如果调用connect失败，进程就休眠一小段时间然后再尝试，每循环一次增加每次尝试的延迟，直到最大延迟为2分钟。

如果套接字描述符处于非阻塞模式下，那么在连接不能马上建立时， connect将会返回-1，并且将errno设为特殊的错误码EINPROGRESS。应用程序可以使用poll或者select来判断文件描述符何时可写。如果可写，连接完成。

函数connect还可以用于无连接的网络服务（SOCK\_DGRAM）。这看起来有点矛盾，实际上却是一个不错的选择。如果在SOCK\_DGRAM套接字上调用connect，所有发送报文的目标地址设为connect调用中所指定的地址，这样每次传送报文时就不需要再提供地址。另外，仅能接收来自指定地址的报文。

服务器调用listen来宣告可以接受连接请求。

|  |
| --- |
| #include <sys/socket.h>  int listen (int sockfd， int backlog);  返回值：若成功则返回0，若出错则返回-1 |

参数backlog提供了一个提示，用于表示该进程所要入队的连接请求数量。其实际值由系统决定，但上限<sys/socket .h>中的SOMAXCONN指定。

一旦队列满，系统会拒绝多余连接请求，所以的值应该基于服务器期望负载和接受连接请求与启动服务的处理能力来选择。

一旦服务器调用了listen,套接字就能接收连接请求。使用函数accept获得连接请求并建立连接。

|  |
| --- |
| #include <sys/socket.h>  int accept (int sockfd, struct sockaddr \*restrict addr, socklen\_t \* restrict len)；  返回值：若成功则返回文件（套接字）描述符。若出错则返回-1 |

函数accept所返回的文件描述符是套接字描述符，该描述符连接到调用connect的客户端。

这个新的套接子描述符和原始套接字(sockfd)具有相同的套接字类型和地址族。传给accept的原始套接字没有关联到这个连接，而是继续保持可用状态并接受其他连接请求。

如果不关心客户端标识，可以将参数addr和设为NULL；否则，在调用accept之前，应将参数addr设为足够大的缓冲区来存放地址，并且将len设为指向代表这个缓冲区大小的整数的指针。返回时，accept会在缓冲区填充客户端的地址并且更新指针所指向的整数为该地址的大小。

如果没有连接请求等待处理，accept会阻塞直到一个请求到来。如果sockfd处于非阻塞模式，accept会返回-1并将errno设置为EAGAIN或EWOULDBLOCK。

如果服务器调用accept并且当前没有连接请求，服务器会阻塞直到一个请求到来。另外，服务器可以使用poll或select来等待一个请求的到来。在这种情况下，一个带等待处理的连接请求套接字会以可读的方式出现。

下面程序显示了一个服务器进程用以分配和初始化套接字的函数。

|  |
| --- |
|  |

我们将会看到，TCP关于地址复用有一些奇怪的规则，导致这个例子并不完备。后面会提供有关这个函数的另一个版本，该版本可以绕过这些规则，解决眼下版本的主要缺陷。

## 12.5数据传输

既然将套接字端点表示为文件描述符，那么只要建立连接，就可以使用read和write来通过套接字通信。回忆前面所讲，通过在connect函数里面设置对方地址，数据报套接字也可以“连接”。在套接字描述符上采用read和write是非常有意义的，因为可以传递套接字描述 符到那些原先设计为处理本地文件的函数。而且可以安排传递套接字描述符到执行程序的子进程，该子进程并不了解套接字。

尽管可以通过read和write交换数据，但这就是这两个函数所能做的一切。如果想指定选项、从多个客户端接收数据包或者发送带外数据，需要采用六个传递数据的套接字函数中的一个。

三个函数用来发送数据，三个用于接受数据。首先，考查用于发送数据的函数。

最简单的是send,它和write很像，但是可以指定标志来改变处理传输数据的方式。

|  |
| --- |
| #include <sys/socket.h>  ssize\_t send (int sockfd, const void \*buf, size\_t nbytes, int flags);  返回值：若成功则返回发送的字节数，若出错则返回-1 |

类似write,使用send时套接字必须已经连接。参数与write中的含义一致。

然而，与write不同的是，send支持第四个参数flags。下表就列出了flags的一些选项和含义。

|  |  |
| --- | --- |
| 标 志 | 描述 |
| MSG\_DONTROUTE | 勿将数据路由出本地网络 |
| MSG\_DONTWAIT | 允许非阻塞操作（等价于使用O\_NONBLOCK） |
| MSG\_E0R | 如果协议支持，此为记录结束 |
| MSG\_OOB | 如果协议支持，发送带外数据 |

如果send成功返回，并不必然表示连接另一端的进程接收数据。所保证的仅是当send成功返回时，数据已经无错误地发送到网络上。

对于支持为报文设限的协议，如果单个报文超过协议所支持的最大尺寸，send失败并将 errno设为EMSGSIZE,对于字节流协议，send会阻塞直到整个数据被传输。

函数sendto和send很类似。区别在于sendto允许在无连接的套接字上指定一个目标地址。

|  |
| --- |
| #include <sys/socket.h>  ssize\_t sendto (int sockfd, const void \*buf, size\_t nbytes, int flags, const struct sockaddr \*destaddr, socklen t destlen)；  返回值：若成功则返回发送的字节数，若出错则返回-1 |

对于面向连接的套接字，目标地址是忽略的，因为目标地址蕴涵在连接中。对于无连接的套接字，不能使用send，除非在调用connect时预先设定了目标地址，或者采用sendto来提供另外一种发送报文方式。

可以使用不止一个的选择来通过套接字发送数据。可以调用带有msghdr结构的sendmsg来指定多重缓冲区传输数据。

|  |
| --- |
| #include <sys/socket.h>  ssize\_t sendmsg(int sockfd, const struct msghdr \*msg, int flags);  返回值；若成功则返回发送的字节数，若出错则返回-1 |

msghdr结构至少应该有如下成员：

|  |
| --- |
| struct msghdr {  void \*msg\_name; /\* optional address \*/  socklen\_t msg\_namelen; /\* size of address \*/  struct iovec \*msg\_iov; /\* scatter/gather array \*/  size\_t msg\_iovlen; /\* # elements in msg\_iov \*/  void \*msg\_control; /\* ancillary data, see below \*/  size\_t msg\_controllen; /\* ancillary data buffer len \*/  int msg\_flags; /\* flags on received message \*/  }; |

函数recv和read很像，但是允许指定选项来控制如何接收数据。

|  |
| --- |
| #include <sys/socket.h>  ssize\_t recv (int sochfd, void \*buf, size\_\_t nbytes, int flags);  返回值：以字节计数的消息长度，若无可用消息或对方已经按序结束则返回0，若出错则返回-1 |

下面是flag参数所有可选的取值列表

|  |  |
| --- | --- |
| 标 志 | 描述 |
| MSG\_OOB | 如果协议支持，接收带外数据 |
| MSG\_PEEK | 返回报文内容而不真正取走报文 |
| MSG\_TRUNC | 即使报文被截断，要求返回的是报文的实际长度 |
| MSG\_WAITALL | 等待直到所有的数据可用（仅SOCK\_STREAM） |

当指定MSG\_PEEK标志时，可以查看下一个要读的数据但不会真正取走。当再次调用read或recv函数时会返回刚才查看的数据。

对于SOCK\_STREAM套接字，接收的数据可以比请求的少。标志MSG\_WAITALL阻止这种行为，除非所需数据全部收到，recv函数才会返回。对于SOCK\_DGRAM和SOCK\_SEQPACKET套接字，MSG\_WAITALL标志没有改变什么行为，因为这些基于报文的套接字类型一次读取就返回整个报文。

如果发送者已经调用shutdown来结束传输，或者网络协议支持默认的顺序关闭并且发送端已经关闭，那么当所有的数据接收完毕后，recv返回0。

如果需要位发送者，可以使用recvfrom来得到数据发送者的源地址。

|  |
| --- |
| #include <sys/socket.h>  ssize\_t recvf rom (int sockfd, void \*restrict buf, sizet len, int flags,  返回值：以字节计数的消息长度，若无可用消息或对方已经按序结束则返回0,若出错则返回-1 |

如果addr非空，它将包含数据发送者的套接字端点地址。当调用recvfrom时，需要设置addrlen参数指向一个包含addr所指的套接字缓冲区字节大小的整数。返回时，该整数设为该地址的实际字节大小。

因为可以获得发送者的地址，recvfrom通常用于无连接套接字。否则，recvfrom等同于recv。

为了将接收到的数据送入多个缓冲区，或者想接收辅助数据，可以使用recvmsg。

|  |
| --- |
| #include <sys/socket.h>  ssize\_t recvmsg (int sockfd, struct msghdr \*msgf int flags)；  返回值：以字节计数的消息长度，若无可用消息或对方已经按序结束则返回0，若出错则返回-1 |

结构msghdr（在sendmsg中见过）被recvmsg用于指定接收数据的输入缓冲区。可以设置参数flag供来改变recvmsg的默认行为。返回时，msghdr结构中的msg\_flags字段被设为所接收数据的各种特征（进入recvmsg时msg\_flags被忽略）。从recvmsg中返回的各种可能值总结在下表中。

|  |  |
| --- | --- |
| 从recvmsg中返回的msg\_flags标志 | |
| 标 志 | 描述 |
| MSG\_CTRUNC | 控制数据被截断 |
| MSG\_DONTWAIT | recvmsg处于非阻塞模式 |
| MSG\_EOR | 接收到记录结束符 |
| MSG\_OOB | 接收到带外数据 |
| MSG\_TRUNC | 一般数据被截断 |

下面的程序显示了一个客户端命令，该命令用于与服务器通信以获得系统命令uptime的输出。该服务称为“remote uptime”（简称为“ruptime”）。

|  |
| --- |
|  |

这个程序连接服务器，读取服务器发送过来的字符串并将其打印到标准输出。既然使用SOCK\_STREAM套接字，就不能保证在一次recv调用中会读取整个字符串，所以需要重复调用直到返回0。

如果服务器支持多重网络接口或多重网络协议，函数getaddrinfo会返回不止一个候选地址。轮流尝试每个地址，当找到一个允许连接到服务的地址时便可停止。

下面的程序是显示服务器程序，用来提供上述uptime命令的客户端程序的输出。

|  |
| --- |
|  |

通过调用gethostname,服务器程序获得主机名字，并查看远程uptime服务地址。可能会有多个地址返回，但简单地选择第一个来建立被动套接字端点。

前面说过采用文件描述符来访问套接字是非常有意义的，因为允许程序对联网环境的网络访问一无所知。下面的服务器程序版本显示了这一点。为了代替从uptime命令中读取输出并发送到客户端，服务器安排uptime命令的标准输出和标准错误替换为连接到客户端的套接字端点。

|  |
| --- |
|  |

以前的方式是采用popen来运行uptime命令，并从连接到命令标准输出的管道读取输出，现在采用fork来创建一个子进程，并使用dup2使子进程的STDIN\_FILENO的副本打开到/dev/null、STDOUT\_FILENO和STDERR\_FILENO打开到套接字端点。当执行uptime时，命令将结果写到标准输出，该标准输出连到套接字，所以数据被送到ruptime客户端命令。

父进程可以安全地关闭连接到客户端的文件描述符，因为子进程仍旧打开着。父进程等待子进程处理完毕，所以子进程不会变成僵死进程。既然运行uptime花费时间不会太长，父进程在接受下一个连接请求之前，可以等待子进程退出。不过，这种策略不适合子进程运行时间比较长的情况。

前面的例子采用面向连接的套接字。但如何选择合适的套接字类型？何时采用面向连接的套接字，何时采用无连接的套接字呢？答案取决于要做的工作以及对错误的容忍程度。

对于无连接套接字，数据包的到来可能已经没有次序，因此当所有的数据不能放在一个包里时，在应用程序里面必须关心包的次序。包的最大尺寸是通信协议的特性。并且对于无连接套接字，包可能丢失。如果应用程序不能容忍这种丢失，必须使用面向连接的套接字。

容忍包丢失意味着两个选择。如果想和对方可靠通信，必须对数据包编号，如果发现包丢失，则要求对方重新传输。既然包可能因延迟而疑似丢失，我们要求重传，但该包却又出现，与重传过来的包重复。因此必须识别重复包，如果出现重复包，则将其丢弃。

另外一个选择是通过让用户再次尝试命令来处理错误。对于简单的应用程序，这就足够，但对于复杂的应用程序，这种处理方式通常不是可行的选择，一般在这种情况下使用面向连接的套接字更为可取。

面向连接的套接字的缺陷在于需要更多的时间和工作来建立一个连接，并且每个连接需要从操作系统中消耗更多的资源。

下面的程序是采用数据报套接字接口的uptime客户端。

|  |
| --- |
|  |

除了为SIGALRM增加一个信号处理程序以外，基于数据报的客户端main函数和面向连接的客户端中的类似。使用alarm函数来避免调用recvfrom时无限期阻塞。

对于面向连接的协议，需要在交换数据前连接服务器。对于服务器来说，到来的连接请求已经足够判断出所需提供给客户端的服务。但是对于基于数据报的协议，需要有一种方法来通知服务器需要它提供服务。本例中，只是简单地给服务器发送1字节的消息。服务器接收后从包中得到地址，并使用这个地址来发送响应消息。如果服务器提供多个服务，可以使用这个请求消息来指示所需要的服务，但既然服务器只做一件事情，1字节消息的内容是无关紧要的。

如果服务器不在运行状态，客户端调用recvfrom便会无限期阻塞。对于面向连接的例子，如果服务器不运行，connect调用会失败。为了避免无限期阻塞，调用recvf rom之前设置警告时钟。

下面的程序是数据报版本的uptime服务器程序。

|  |
| --- |
|  |

服务器程序在recvfrom中阻塞等待服务请求。当一个请求到达时，保存请求者地址并使用popen来运行uptime命令。采用sendto函数将输出发送到客户端，其目标地址就设为刚才的请求者地址。

## 12.6套接字选项

套接字机制提供两个套接字选项接口来控制套接字行为。一个接口用来设置选项，另一个接口允许查询一个选项的状态。可以获取或设置三种选项：

* 通用选项，工作在所有套接字类型上。
* 在套接字层次管理的选项，但是依赖于下层协议的支持。
* 特定于某协议的选项，为每个协议所独有。

可以采用setsockopt函数来设置套接字选项。

|  |
| --- |
| #include <sys/socket.h>  int setsockopt (int sockfd, int level, int option, const void \*val,  socklen—t len);  返回值：若成功则返回0，若出错则返回-1 |

参数level标识了选项应用的协议。如果选项是通用的套接字层选项，level设置成S0L\_S0CKET。否则，level设置成控制这个选项的协议号。例如，对于TCP选项，这是IPPR0T0\_TCP，对于IP 选项，这是IPPR0T0\_IP。下表总结了通用套接字层的选项。

|  |  |  |
| --- | --- | --- |
| 选项 | val类型 | 描 述 |
| SO\_ACCEPTCONN | int | 返回信息指示该套接字是否能监听（仅getsockopt） |
| S0\_BROADCAST | int | 如果\*val非零，广播数据包 |
| S0\_DEBUG | int | 如果\*val非零，启用网络驱动调试功能 |
| SO\_DONTROUTE | int | 如果\*val非零，绕过通常路由 |
| SO\_ERROR | int | 返回挂起的套接字错误并清除（仅getsockopt） |
| SO\_KEEPALIVE | int | 如果\*val非零，启用周期性keep-alive消息 |
| SO\_LINGER | struct linger | 当有未发消息并且套接字关闭时，延迟时间 |
| SO\_OOBINLINE | int | 如果\*val非零，将带外数据放在普通数据中 |
| SO\_RCVBUF | int | 以字节为单位的接收缓冲区大小 |
| SO\_RCVLOWAT | int | 接收调用中返回的以字节为单位的最小数据量 |
| SO\_RCVTIMEO | struct timeval | 套接字接收调用的超时值 |
| SO\_REUSEADDR | int | 如果\*val非零，重用bind中的地址 |
| SO\_SNDBUF | int | 以字节为单位的发送缓冲区大小 |
| SO\_SNDLOWAT | int | 发送调用中以字节为单位的发送的最小数据量 |
| SO\_SNDTIMEO | struct timeval | 套接字发送调用的超时值 |
| SO\_TYPE | int | 标识套接字类型（仅getsockopt） |

参数val根据选项的不同指向一个数据结构或者一个整数。一些选项是on/off开关。如果整数非零，那么选项被启用。如果整数为零，那么该选项被禁止。参数len指定了val指向的对象 的大小。

可以使用getsockopt函数来发现选项的当前值。

|  |
| --- |
| #include <sys/socket.h>  int getsockopt (int sockfd, int level, int option, void \*restrict val,  socklen\_t \* restrict lenp);  返回值：若成功则返回0，若出错则返回-1 |

注意到参数lenp是一个指向整数的指针。在调用getsockopt之前，设置该整数为复制选项缓冲区的大小。如果实际的尺寸大于此值，选项会被截断而不报错；如果实际尺寸正好等于或者小于此值，那么返回时将此值更新为实际尺寸。

当服务器终止并尝试立即重启时，上面的程序可能不会正常工作。除非超时（这个通常约为几分钟），通常TCP的实现不允许绑定同一个地址。幸运的是套接字选项 SO\_REUSEADDR允许越过这个限制，新的程序改进如下：

|  |
| --- |
|  |

为了启用SO\_REUSEADDR选项，在setsockopt中val的参数设置为一个非零整数的地址。设置len参数为val所指的对象的大小。 □

## 12.7 带外数据

带外数据（Out-of-band data）是一些通信协议所支持的可选特征，允许更高优先级的数据比普通数据优先传输。即使传输队列已经有数据，带外数据先行传输。TCP支持带外数据，但是UDP不支持。套接字接口对带外数据的支持，很大程度受TCP带外数据具体实现的影响。

TCP将带外数据称为“紧急”数据（“urgent” data）。TCP仅支持一个字节的紧急数据，但是允许紧急数据在普通数据传递机制数据流之外传输。为了产生紧急数据，在三个send函数中任何一个指定标志MSG\_OOB。如果带MSG\_OOB标志传输字节超过一个时，最后一个字节被看作紧急数据字节。

如果安排发生套接字信号，当接收到紧急数据时，那么发送信号SIGURG。可以在fcntl中使用F\_SETOWN命令来设置一个套接字的所有权。如果 fcntl中第三个参数为正值，那么指定了进程ID，如果为非-1的负值，那么代表了进程组ID。因此，通过调用以下函数，可以安排进程接收一个套接字的信号。

fcntl(sockfd, F\_SETOWN, pid);

F\_GETOWN命令可以用来获得当前套接字所有权。对于F\_SETOWN命令，一个负值代表一个进程组ID，一个正值代表进程ID。因此，调用

owner = fcntl(sockfd, F\_GETOWN, 0);

返回值owner，如果owner为正值，则owner等于配置为接受套接字信号的进程ID，如果 owner为负值，则其绝对值为接受套接字信号的进程组ID。

TCP支持紧急标记（urgent mark）的概念：在普通数据流中紧急数据所在的位置。如果采用套接字选项SO\_OOBINLINE，那么可以在普通数据中接收紧急数据。为帮助判断是否接收到紧急标记，可以使用函数sockatmark。

|  |
| --- |
| #include <sys/socket.h>  int sockatmark (int sockfd);  返回值：若在标记处则返回1，若没有在标记处则返回0，若出错则返回-1 |

当下一个要读的字节在紧急标志所标识的位置时，sockatmark返回1。

当带外数据出现在套接字读取队列时，select函数会返回一个文件描述符并 且拥有一个异常状态挂起。可以在普通数据流上接受紧急数据，或者在某个recv函数中采用MSG\_OOB标志在其他队列数据之前接收紧急数据。TCP队列仅有一字节的紧急数据，如果在接收当前的紧急数据字节之前又有新的紧急数据到来，那么当前的字节会被丢弃。

## 12.8 非阻塞和异步I/O

通常，recv函数没有数据可用时会阻塞等待。同样地，当套接字输出队列没有足够空间来发送消息时函数send会阻塞。在套接字非阻塞模式下，行为会改变。在这种情况下，这些函数不会阻塞而是失败，设置errno为EWOULDBLOCK或者EAGAIN。当这些发生时，可以使用poll或select来判断何时能接收或者传输数据。

在LINUX/UNIX中，其实时扩展包含对通用异步I/O机制的支持。套接字机制有自己的方式来处理异步I/O，但是没有标准化。一些文献把经典的基于套接字的异步I/O机制称为“基于信号的I/O”，以区别于实时扩展中的异步I/O机制。

在基于套接字的异步I/O中，当能够从套接字中读取数据，或者套接字写队列中的空间变得可用时，可以安排发送信号SIGIO。通过两个步骤来使用异步I/O：

* 建立套接字拥有者关系，信号可以被传送到合适的进程。
* 通知套接字当I/O操作不会阻塞时发信号告知。

可以使用三种方式来完成第一个步骤：

* 在f cntl使用F\_SETOWN命令。
* 在ioctl中使用FIOSETOWN命令。
* 在ioctl中使用SIOCSPGRP命令。

要完成第二个步骤，有两个选择：

* 在fcntl中使用F\_SETFL命令并且启用文件标志O\_ASYNC。
* 在ioctl中使用FIOASYNC。

虽然有不少选择，但不是普遍得到支持。

## 12.9小结

在本章中，考查了另外一种机制，这种机制允许一个进程与另外一个进程通信，无论是不同的机器上还是同一机器中。讨论了套接字端点如何命名，在连接服务器时，如何发现所要用的地址。

我们给出了采用无连接的套接字（例如，基于数据报）和面向连接套接字的客户端和服务器的例子。简要讨论了异步和非阻塞的套接字I/O，以及用于管理套接字选项的接口。